

# ECLIPSE AND EMBEDDED LINUX DEVELOPERS: WHAT IT CAN AND CANNOT DO FOR YOU

Anna Dushistova  
[anna.dushistova@gmail.com](mailto:anna.dushistova@gmail.com)



# WHY IDE?

From wikipedia:

“...IDEs are designed to maximize programmer productivity by providing *tightly-knit components* with *similar user interfaces*. This should mean that the programmer has *much less mode switching* to do than when using discrete development programs...”



# TYPICAL CONTENT OF AN IDE

- a source code editor
- a compiler and / or an interpreter
- build automation tools
- a debugger
- SCM integration



# ABOUT ECLIPSE

- What is it?

- Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle.

- History of Eclipse

- Started in November 2001 by Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain.
- On Feb 2, 2004 the Eclipse Board of Stewards announced Eclipse's reorganization into a not-for-profit corporation. The founding Strategic Developers and Strategic Consumers were Ericsson, HP, IBM, Intel, Monta Vista Software, QNX, SAP and Serena Software.



# WHAT CAN ECLIPSE OFFER?

- C/C++ Development Tooling (CDT) for edit/compile/debug cycle
- Target Management (TM) or Target Communication Framework (TCF) with Target Explorer (TE) for working with remote hosts
- CVS, SVN and Git integration
- Linux Tools for various Linux specific performance monitoring tools (gprof, lttng, oprofile, systemTap)



# SOME COMPANIES BEHIND THESE PROJECTS

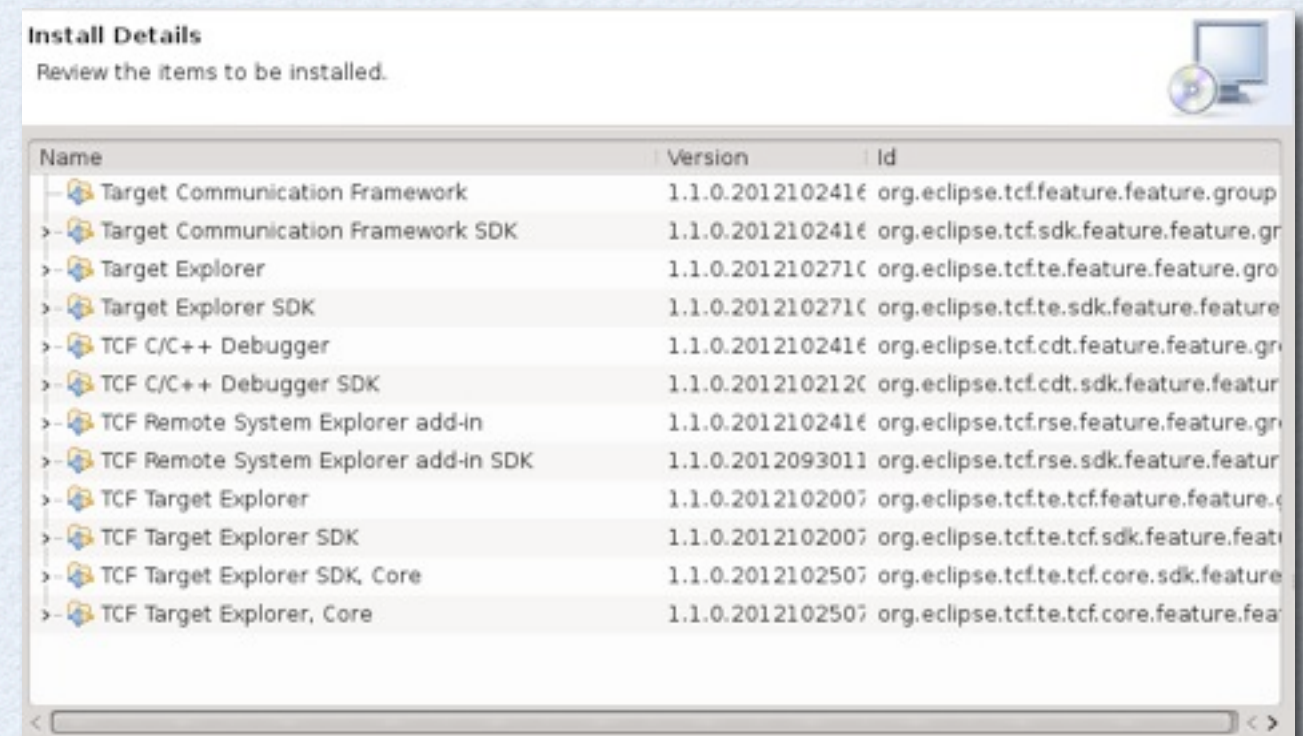
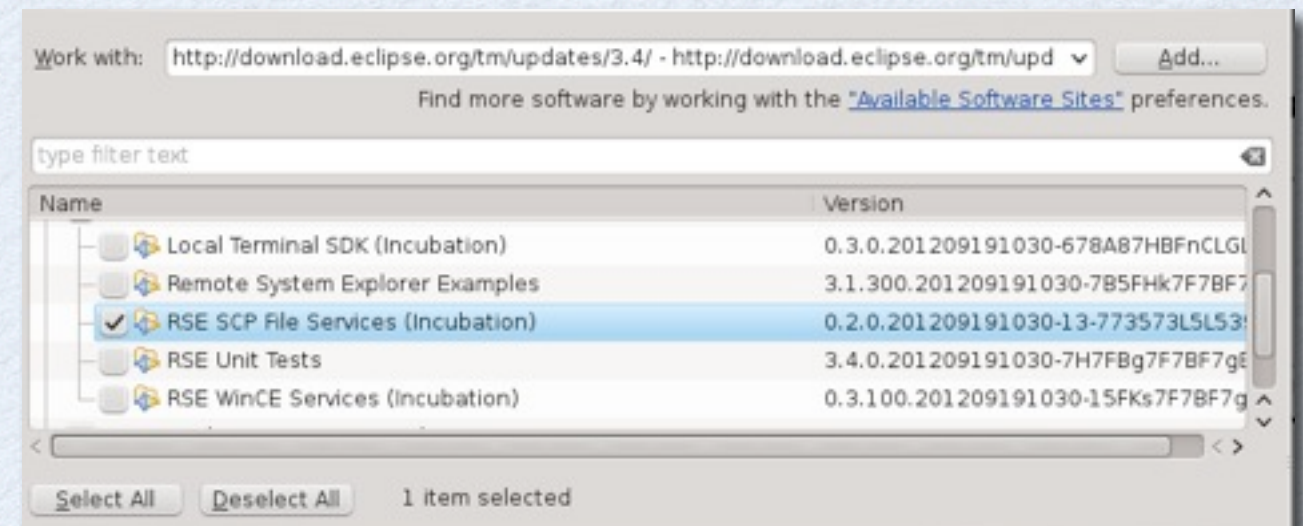




# BUT...

- Is it possible to use Eclipse with standard versions of these plugins for embedded Linux development?

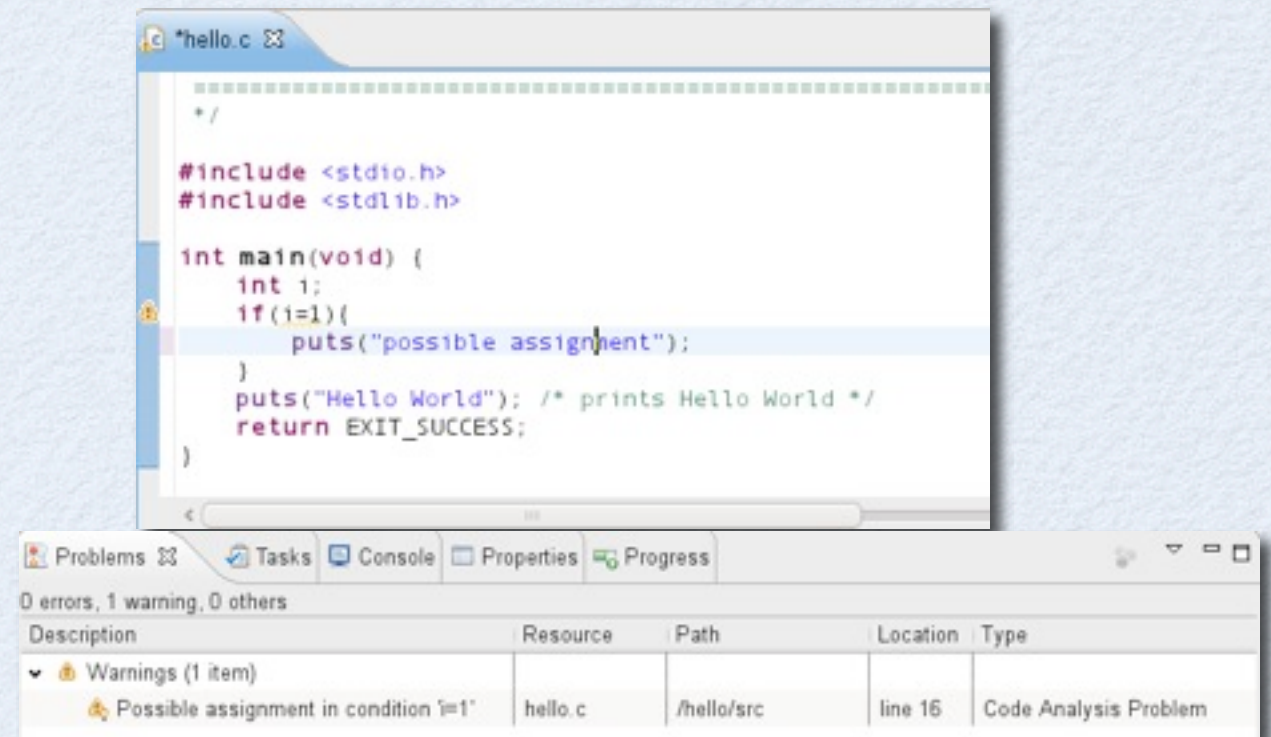
- Let's start with Eclipse IDE for C/C++ Developers package (<http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/junosr1>)
- Install additional Target Management feature(SCP file subsystem) (update site: <http://download.eclipse.org/tm/updates/3.4/>)
- Install latest version of Target Explorer (update site: <http://download.eclipse.org/tools/tcf/builds/development/nightly/>)





# WORKING WITH C/C++ CODE

- Code Analysis Framework “Codan”
  - Now supports external code analysis tools like cppcheck:
    - External tools can be configured using Codan’s preference page
    - External tools are invoked automatically when a C/C++ file is saved
    - The output of these tools can be displayed as editor markers





# BUILDING YOUR APPLICATION

- Works perfectly for a local toolchain out of the box!
- However, when it comes to using cross-compilation tools...



# BUILDSYSTEMS SUPPORTED IN CDT

CDT Managed Make	Standard Make	Autotools
Makefiles are generated by CDT	You write your own makefiles	Build scripts are generated by automake tools
Doesn't have support outside CDT	Standard Linux way of software development	Standard Linux way of software development
Requires complex toolchain definition for CDT, has a lot of limitations	Needs only a set of environment variables	Needs only a set of environment variables and proper arguments to pass to autoconf



# DEALING WITH CDT'S MANAGED MAKE

- How can we integrate CDT with our cross-compilation tools?
- Write a plugin for Eclipse that adds support for it
- Use “Cross GCC”
- Redefine settings for each project manually



# DEFINING YOUR OWN TOOLCHAIN FOR CDT

- If you don't need any special settings, just extend existing definitions for "Linux GCC"
- The more special your needs, the more changes you will have to make
- Changes are not obvious and require some Eclipse experience

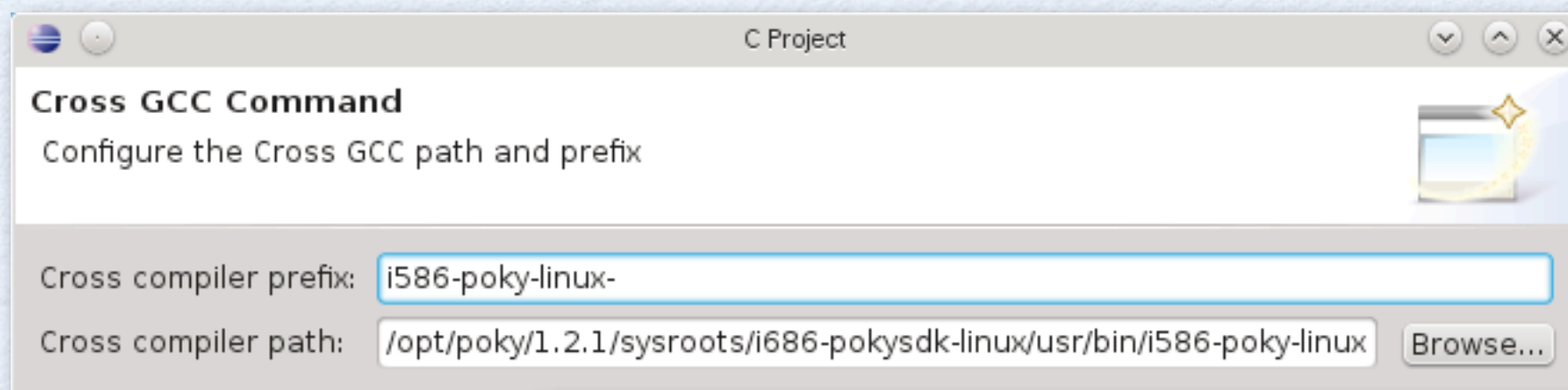
```
name="Anna's GCC" osList="all">
  <targetPlatform
    id="cdttest309024.gnu.platform.base"
    name="Debug"
    binaryParser="org.eclipse.cdt.core.ELF"
    osList="linux,hpux,aix,qnx"
    archList="all">
  </targetPlatform>
  <builder
    superClass="cdt.managedbuild.target.gnu.builder"
    id="cdt.test309024.builder.base">
  </builder>

  <tool command="$(CXX)"
    id="cdt.test309024.cpp.compiler" isAbstract="false"
    name="Anna's GNU G++ Compiler" superClass="cdt.managedbuild.tool.gnu.cpp.compiler">
  </tool>
```



# USING "CROSS GCC"

- Requires just two settings:
  - path
  - prefix(make sure you add dash in the end)



```
CDT Build Console [test1]
22:09:02 **** Build of configuration Debug for project test1 ****
make all
Building file: ../src/test1.c
Invoking: Cross GCC Compiler
i586-poky-linux-gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/test1.d" -MT"src/test1.d" -o "src/
test1.o" "../src/test1.c"
Finished building: ../src/test1.c

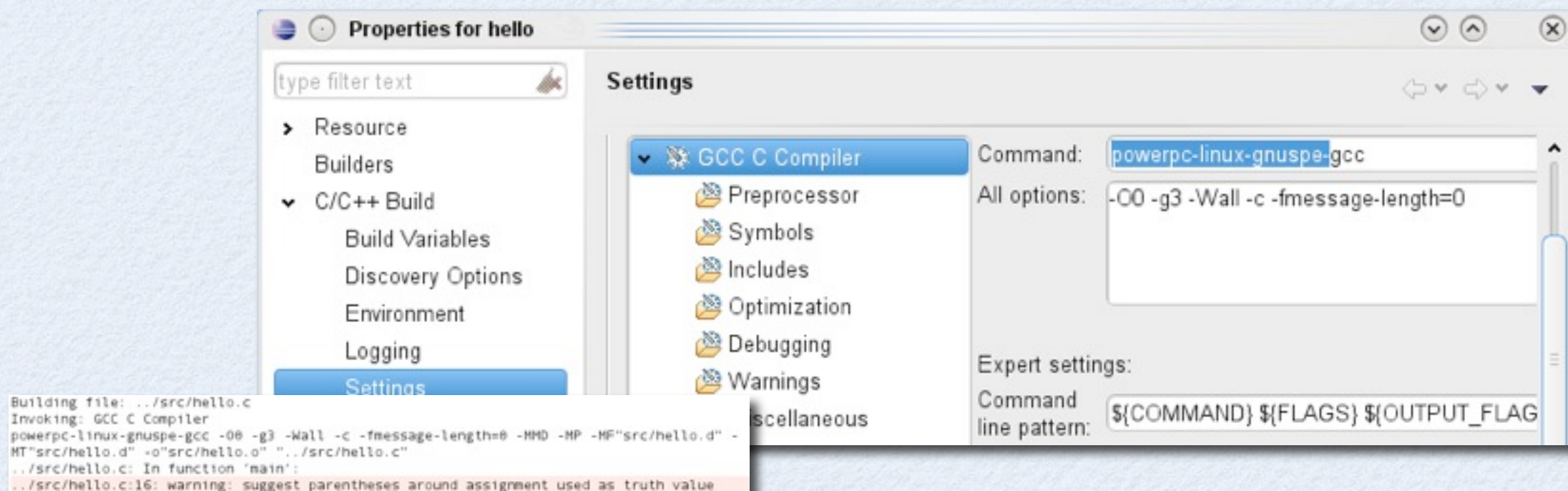
Building target: test1
Invoking: Cross GCC Linker
i586-poky-linux-gcc -o "test1" ./src/test1.o |
Finished building target: test1

22:09:03 Build Finished (took 604ms)
```



# REDEFINING SETTINGS MANUALLY

- Must be done for each configuration in each project in use
- Requires the following modifications of project properties:
  - changing gcc to your <arch>-gcc in Settings for each tool
  - Modifying PATH variable(adding your cross-toolchain path)





# WORKING WITH REMOTE SYSTEMS

Since summer 2012 there are two options:

- Remote System Explorer (RSE) provided by Target Management project
- Target Explorer (TE) coming from Target Communication Framework project



# REMOTE SYSTEMS EXPLORER VS. TARGET EXPLORER

Remote Systems Explorer	Target Explorer
Mature, stable project (1.0 in November 2006)	1.0 version was released in June 2012, designed to be a lightweight successor of RSE for embedded
Transport protocol agnostic	Centered around Target Communication Framework(TCF) protocol and assumes there is a TCF agent on the remote target
Does not support auto-discovery	Supports auto-discovery of connections and services



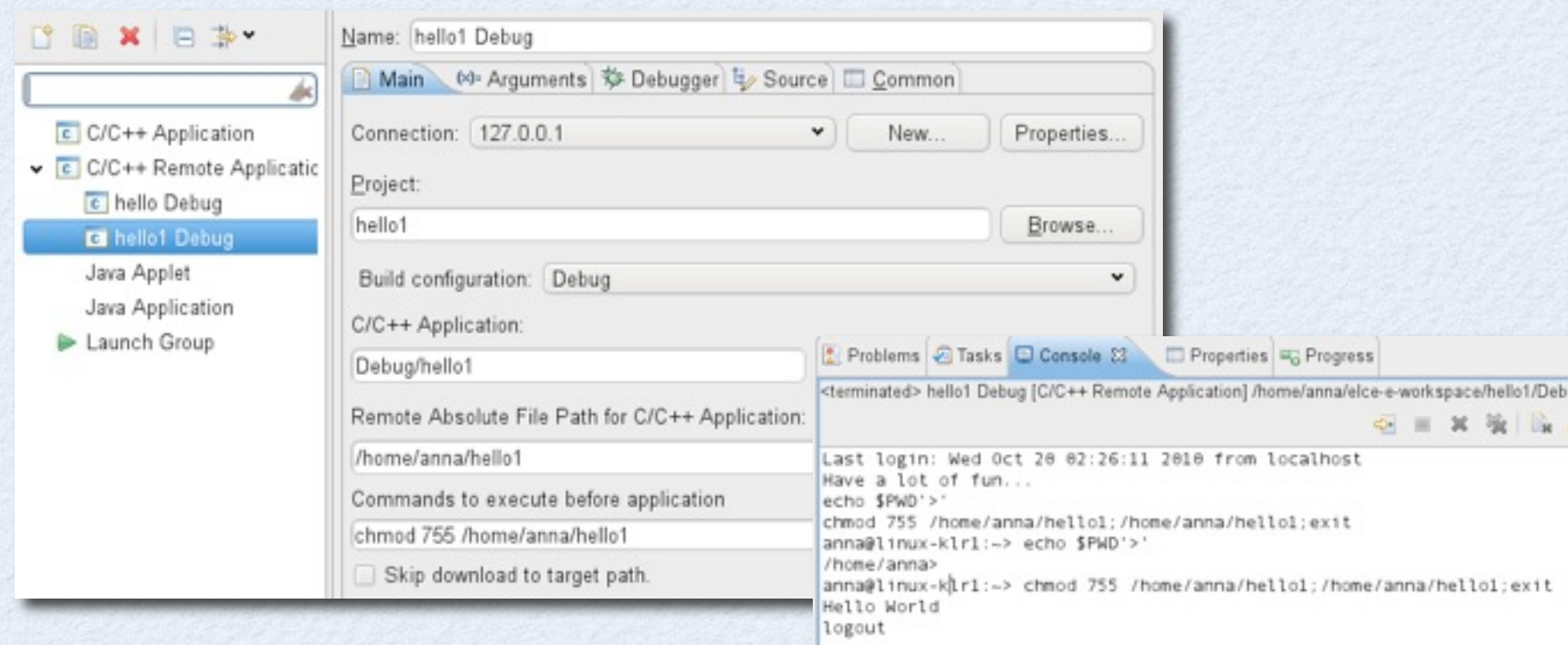
# TARGET COMMUNICATION FRAMEWORK

- Vendor-neutral, lightweight, extensible network protocol mainly for communicating with embedded systems (targets)
- Has a transport-agnostic channel abstraction
- Supports auto-discovery of targets and services



# RUNNING YOUR APPLICATIONS USING RSE

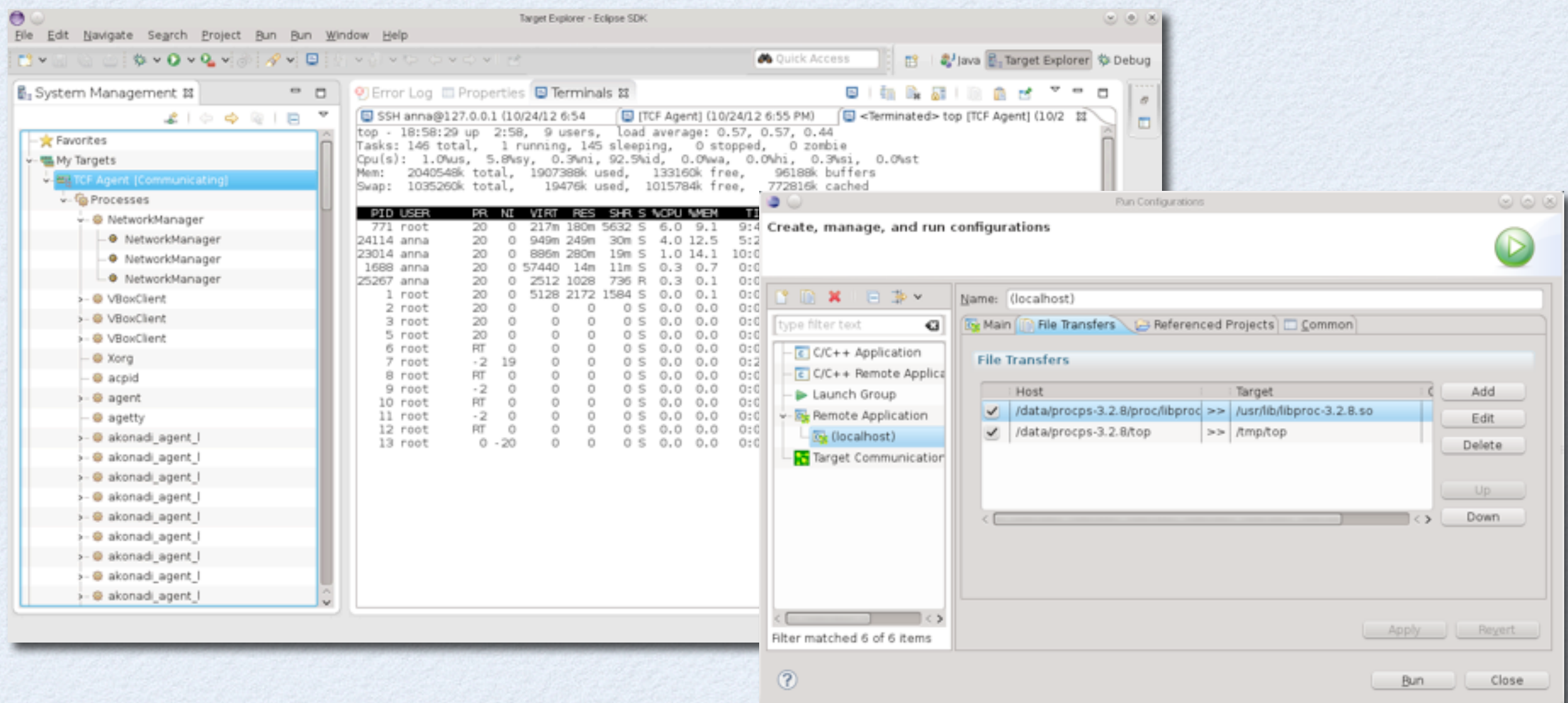
- Use Remote Systems Explorer to copy your application on to the target and then a terminal to launch it
- Use Remote Launch to launch your application from Eclipse
- Transferring more than one file(e.g. application and library) is not supported in the Remote Launch





# RUNNING YOUR APPLICATIONS USING TE

- Remote Launch from Target Explorer allows you to transfer all necessary files and run your application in one step
- Output is shown in ANSI terminal





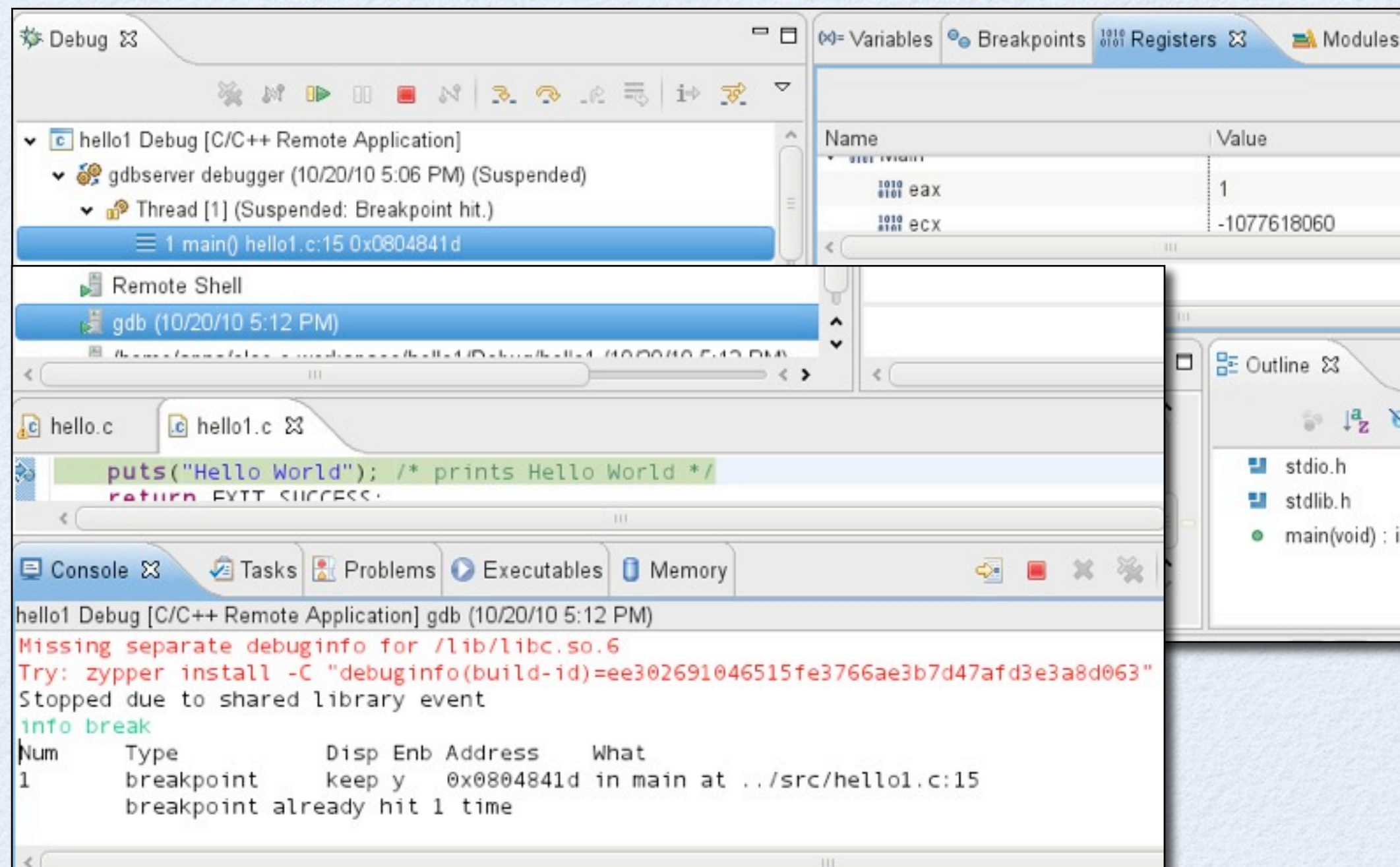
# DEBUGGING YOUR APPLICATIONS

Options offered:

- GDB/GDB server solution
  - Automatic remote debugger (launches gdbserver on remote side automatically, doesn't support attaching to a running process)
    - Current implementation uses RSE
    - TE-based solution pending review
  - Manual remote debugger (you have to launch gdbserver manually, supports attach)
- TCF-agent-based debugger
  - only x86 debugging support in the open source TCF agent at the moment



# DEBUGGING YOUR APPLICATIONS-GDB





# DEBUGGING YOUR APPLICATIONS-TCF

Debug - procs-test/top.c - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access C/C++ Target Explorer Remote System Explorer LTng Kernel Debug

Debug P1386.1386 (Breakpoint: Trace/breakpoint trap)

- 0x080507d6 [top] summary\_show(): top.c, line 2951
- 0x0805363d [top] frame\_make(): top.c, line 3304
- 0x08053950 [top] main(): top.c, line 3369
- 0x4617c908 [libc-2.13.so] \_\_libc\_start\_main()

Variables Breakpoints Registers Modules

- top.c [line: 1981]
- top.c [line: 2947]

top.c

```
static CPU_t *smppcpu = NULL;

// whoa first time, gotta' prime the pump...
if (!p_table) {
    p_table = procs_refresh(NULL, Frames_libflags);
    putp(Cap_clr_scr);
    putp(Cap_rmam);
#ifdef PROF
    // sleep for half a second
    tv.tv_sec = 0;
    tv.tv_usec = 500000;

```

Outline

- sys/ioctl.h
- sys/resource.h
- sys/time.h
- sys/types.h
- sys/stat.h
- ctype.h

Console Tasks Problems Executables Terminals Memory

/tmp/top [TCF Agent] (11/1/12 9:21 PM)

8	root	RT	0	0	0	S	0.0	0.0	0:00.00	migration/0
9	root	0	-20	0	0	S	0.0	0.0	0:00.00	cpuset
10	root	0	-20	0	0	S	0.0	0.0	0:00.00	khelper
11	root	0	-20	0	0	S	0.0	0.0	0:00.00	netns
12	root	20	0	0	0	S	0.0	0.0	0:00.28	kworker/u:1
312	root	0	-20	0	0	S	0.0	0.0	0:00.00	kblockd



# LINUX MEASUREMENT AND DIAGNOSTIC TOOLS

- Profiling
  - oprofile: remote launching is not supported, only local
  - perf: remote launch exists, but uses Remote Development Tools instead of RSE or TE
  - valgrind: only local launch is supported
- Tracing
  - Supports LTTng 2.0, uses RSE for connecting to remote targets



# ECLIPSE LTTNG INTEGRATION

The screenshot shows the Eclipse IDE with the LTTng Kernel - TCF Agent interface. The interface is divided into several panes:

- Project Explorer:** Shows the project structure for 'test-lttng', including 'Experiments [0]', 'Traces [1]', and 'kernel'.
- Control Flow:** A table showing the execution flow of processes. The table has columns: Process, TID, PTID, Birth time, and Trace.
- Events - kernel:** A table showing kernel events. The table has columns: Timestamp, Channel, Event Type, and Content.
- Histogram:** A graph showing the frequency of events over time. It includes a 'Current Event (sec)' field and a 'Window Span (sec)' field.

**Control Flow Table:**

Process	TID	PTID	Birth time	Trace
klogd	1335		01:31:07.639305475	kernel
tcf-agent	1360		01:31:06.885590104	kernel
tcf-agent	1364		01:31:06.850355753	kernel
sh	1367		01:31:35.433491413	kernel
sh	1447	1367	01:31:36.170104387	kernel
top	1447	1367	01:31:36.170104387	kernel
sh	1448	1367	01:31:54.201420436	kernel
lttng	1448	1367	01:31:54.201420436	kernel

**Events - kernel Table:**

Timestamp	Channel	Event Type	Content
<srch>	<srch>	<srch>	<srch>
01:31:06.318539254	channel0_0	hrtimer_cancel	hrtimer=3354324576
01:31:06.318669434	channel0_0	hrtimer_expire_entry	function=3238521152, now=1739960029161, hrtimer=3354324576
01:31:06.318770844	channel0_0	softirq_raise	vec=1
01:31:06.318822534	channel0_0	sched_stat_runtime	tid=1396, comm=lttng-sessiond, vruntime=36099721672, runtime=2018859

**Histogram:**

Current Event (sec): 1351413066.318539254  
Window Span (sec): 0.099999542

The histogram shows a high frequency of events, with a peak around 1351413136.573146690.



# EMBEDDED DISTRIBUTION TOOLS

- Yocto Eclipse plugin (learn more at the Yocto Developer day on Thursday!)
- Buildroot Eclipse plugin (see <http://www.eclipsecon.org/europe2012/sites/eclipsecon.org/europe2012/files/Buildroot.pdf> for details)



# THE RESULT OF OUR EXPERIMENT

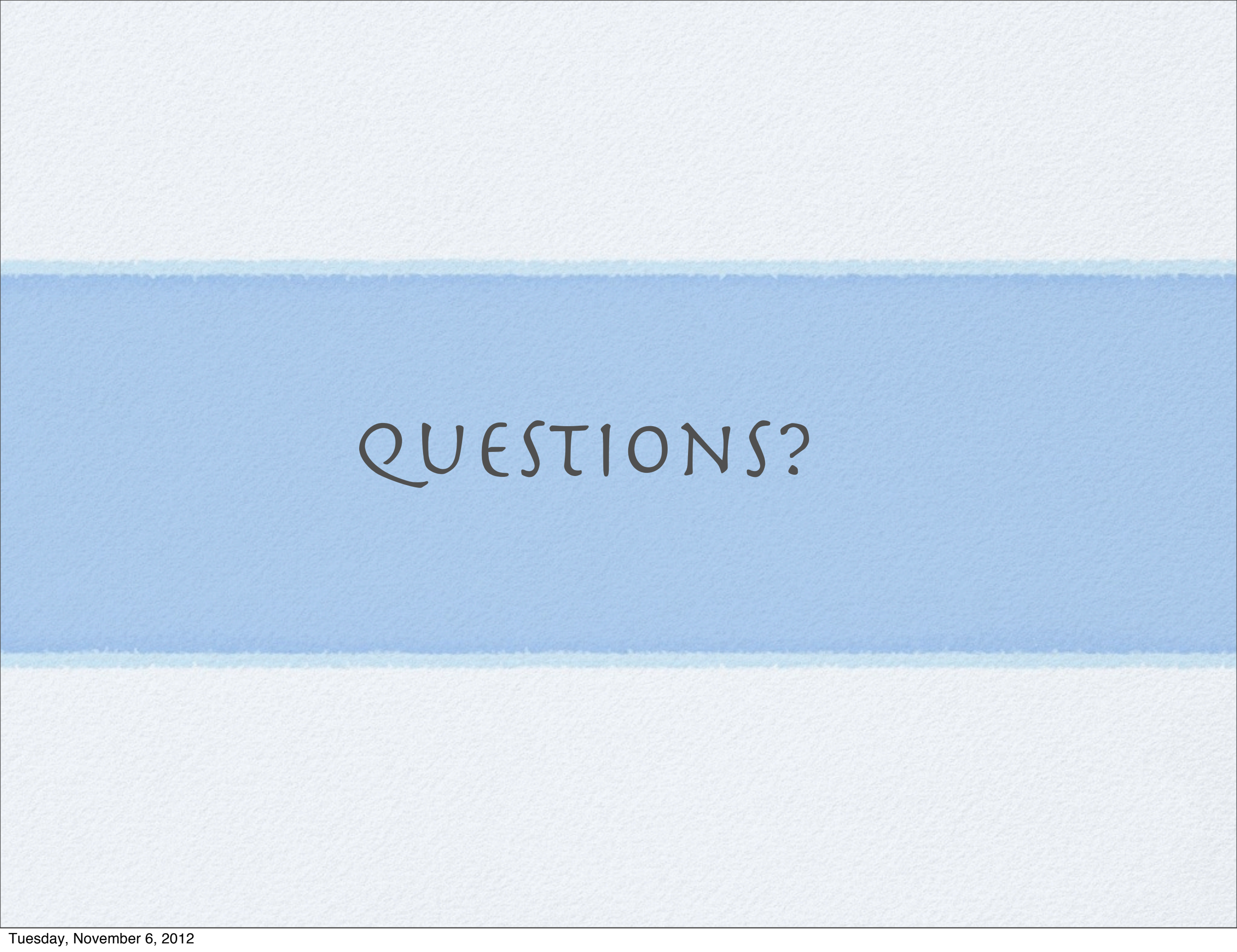
- Technology is there!
- But:
  - almost every piece requires non-trivial actions to make it work the way we need
  - some cannot be used for cross development
  - pieces do not integrate very well
- Overall, the set of plugins and features we installed can hardly be called an IDE.



# DEMO SETUP

- Host:
  - Yocto standalone cross toolchain 1.2.1 for x86(32 bit)
  - Eclipse IDE for C/C++ Developers package Juno SR1
- Target(qemu x86):
  - Stable kernel 3.6.2 from kernel.org
  - Yocto core-image-sato-sdk filesystem image for x86 qemu
  - Lttng 2.0 packages(lttng-modules-2.0.3, lttng-tools-2.0.1, lttng-ust-2.0.3 and userspace-rcu-0.7.3)
  - Latest TCF agent





QUESTIONS?



# ACKNOWLEDGEMENTS

- Uwe Stieber
- Laurette Wharton
- Stanislav Yakovlev
- Eclipse committers and contributors
- Yocto project





THANK YOU!